Pattern based diving heuristics for a two-dimensional guillotine cutting-stock problem with leftovers

François Clautiaux, Ruslan Sadykov, François Vanderbeck, Quentin Viaud

Pattern based diving heuristics for a two-dimensional guillotine cutting-stock problem with leftovers

François Clautiaux[a,b,*], Ruslan Sadykov[b,a], François Vanderbeck[a,b], Quentin Viaud[a,b]

[a] *IMB, Université de Bordeaux, 351 cours de la Libération, 33405 Talence, France*
[b] *INRIA Bordeaux - Sud-Ouest, 200 avenue de la Vieille Tour, 33405 Talence, France*

Abstract

We consider a variant of two-dimensional guillotine cutting-stock problem that arises when different bills of order (or batches) are considered consecutively. The raw material leftover of the last cutting pattern is not counted as waste as it can be reused for cutting the next batch. The objective is thus to maximize the length of the leftover. We propose a diving heuristic based on a Dantzig-Wolfe reformulation solved by column generation in which the pricing problem is solved using dynamic programming (DP). This DP generates so-called non-proper columns, *i.e.* cutting patterns that cannot participate in a feasible integer solution of the problem. We show how to adapt the standard diving heuristic to this "non-proper" case while keeping its effectiveness. We also introduce the partial enumeration technique, which is designed to reduce the number of non-proper patterns in the solution space of the dynamic program. This technique helps to strengthen the lower bounds obtained by column generation and improve the quality of solutions found by the diving heuristic. Computational results are reported and compared on classical benchmarks from the literature as well as on new instances inspired from industrial data. According to these results, proposed diving algorithms outperform constructive and evolutionary heuristics.

Keywords: Cutting and Packing, Dynamic Programming, Diving Heuristic

1. Introduction

The two-dimensional guillotine cutting-stock problem is a classical problem that occurs in industry when one has to cut rectangular pieces from identical large objects (plates) in such a way that the number of plates used is minimum. In wood, glass, steel, or paper industries, most cutting devices impose so-called

*Corresponding author. Tel.: +33 5 40 00 21 37
Email addresses: francois.clautiaux@u-bordeaux.fr (François Clautiaux), ruslan.sadykov@inria.fr (Ruslan Sadykov), francois.vanderbeck@u-bordeaux.fr (François Vanderbeck), quentin.viaud@u-bordeaux.fr (Quentin Viaud)

guillotine cuts. These cuts are made in a straight line from one border of the plate to the other. Motivated by an industrial application, we restrict our attention to cutting patterns that emanate from a 4-stage guillotine-cut process: *i.e.* each cut piece can be obtained trough the process of applying at most 4 guillotine cuts.

In our problem, we are given an item set $\mathcal{I}$. Each item $i \in \mathcal{I}$ is a rectangle with dimensions $w_i \times h_i$, called width and height, and has a demand (or number of copies) equal to d_i. Each item i can be rotated, in which case, its dimensions become $h_i \times w_i$. To cut these items, an unlimited number of identical plates (bins) of dimension $W \times H$ are available. In the remainder of the paper, we assume that all input data are integer.

In practice, set $\mathcal{I}$ is partitioned into distinct batches $\{\mathcal{I}_1, \ldots, \mathcal{I}_B\}$, which have to be processed independently. This saves operational costs, but leads to a larger amount of leftovers (also called waste). To reduce the loss of material, it is allowed to use the leftover from the last plate of the previous batch to initialize the current batch. A leftover is the result of a first stage cut, *i.e.* it is a rectangular piece of height H and width $W' < W$. Therefore, for a given batch, a solution consists of a number of cutting patterns for standard plates, plus a first-plate cutting pattern that has a smaller width, and a last-plate cutting pattern for which the objective is to minimize the width. We call this problem *two-dimensional guillotine cutting-stock problems with leftover* (2DGCSPL). A solution is depicted in Figure 1.

The problem is NP-hard, since it generalizes cutting-stock. It is also combinatorially complex and the practical instances that we need to solve are large: 10 to 15 batches each having about 150 different items, for which the total demand can be as large as 400 copies, and the plates have a large size (6000×3000) relative to the item size. Hence, we consider only so-called *restricted cuts*, *i.e.* cuts of length equal to an item width w_i or an item height h_i. Special requirement related to restricted cuts is that one of the two sub-plates is immediately initialized with an item of width or height equal to the length of the performed cut. Our initial experimental observations as well as those reported by Furini et al. [12] for a related problem confirm that using only restricted cuts does not deteriorate the solutions in most cases.

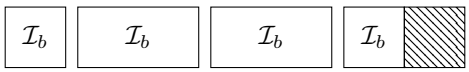

Figure 1: Representation of a solution for a batch $\mathcal{I}_b$. The first plate has a smaller width, since it is the leftover from the previous batch. The right-hand part of the last plate will be re-used in the next batch.

The paper is organised as follows. We first give an overview of the literature on related problems in Section 2. Then in Section 3, we recall how it can be modelled by an exponentially large ILP model in which each variable represents a cutting pattern, and how the pricing subproblem can be solved by dynamic programming. Constructive and evolutionary heuristics inspired from the literature for the 2DGCSPL are discussed in Section 4. In Section 5 we present our

pattern based diving heuristics for the 2DGCSPL. To obtain better solutions by these diving heuristics, we also propose a partial enumeration technique that is embedded in the dynamic program for solving the pricing subproblem. In Section 6, we report results of computational experiments in which we compare different heuristic algorithms on classical data sets and real-life instances. Conclusions are drawn in Section 7.

2. Literature review

Cutting-stock problems have drawn a large attention because of their significance for the industry, and their theoretical and practical difficulty. When all demands are unitary ($d_i = 1, \forall i \in \mathcal{I}$), the problem is also referred in the literature as the bin-packing problem. From a theoretical point of view, one can reformulate a bin-packing problem as a cutting-stock problem, therefore through the rest of this paper, we will use both names.

The first study on two-dimensional packing problems appeared in Gilmore and Gomory [13]. Therein, the approach is to formulate the two-dimensional bin-packing problem (2BP) using Linear Programming (LP) and solve it using column generation. Indeed, the Dantzig-Wolfe decomposition of the 2BP gives a nice reformulation with a covering problem as the master and a two-dimensional knapsack as pricing problem. In Gilmore and Gomory [13] the two-dimensional knapsack pricing problem is tackled by dynamic programming. For the three-stage version of the problem, Vanderbeck [28] used a nested decomposition, solving the pricing problem using Dantzig-Wolfe reformulation. When the pricing problem is too hard to solve, a level approach can be used instead as outlined by Puchinger and Raidl [22]. It combines different methods such as heuristics, meta-heuristics or ILP models. Methods are called one after another and when an acceptable solution is found, it is returned. Another way to solve 2BP was proposed by Silva et al. [26]. It uses an enumeration of the possible patterns occuring during the cutting process. These patterns are fed into a direct Integer Linear Programming (ILP) formulation, which is solved with a general purpose ILP solver. Another approach based on a large ILP model has been developed by Macedo et al. [17]. Authors used the original arc-flow model of Valério de Carvalho [27] for the one-dimensional bin-packing problem (1BP) and extended it to the 2BP.

Most exact methods for the 2BP are not able to scale up to larger instances. This comes from the pseudo-polynomial nature of the models used (either in an ILP formulation, or in the subproblem). To overcome this difficulty, heuristics are often preferred to exact methods. New advanced heuristics such as the agent-based heuristic of Polyakovsky and M'Hallah [21] or the repairing heuristic of Fleszar [10] have been shown to be effective.

Column generation or pattern based heuristics have also been proposed. Alvarez-Valdes et al. [1] used a simple and a more elaborated ad-hoc rounding heuristics. The same authors also used a truncated restricted master heuristic. Furini et al. [11] employed pure diving, restricted master, and diving with sub-MIPing heuristics, if one uses the terminology defined by Sadykov et al. [25].

Since the subproblem is time consuming, the previous authors used heuristics to solve it most of the time. Finally, Cintra et al. [5] worked with column generation approach with non-proper columns. They solved the residual problem obtained after initial rounding using an ad-hoc heuristic.

The main bottleneck of the Dantzig-Wolfe reformulation of the 2BP is the two-dimensional knapsack problem (2KP) pricing subproblem. When the pricing problem is unbounded (*i.e.* when there are no upper bounds on item productions), dynamic programming (DP) is a natural choice to handle pricing. Recurrence relations were initially introduced by Beasley [3]. Russo et al. [24] extended them to tackle huge pricing problems. However it is not straightforward to extend DP to the bounded case, or to a larger number of stages. In Dolatabadi et al. [7], the authors proposed a DP method for the case where the number of cutting stages is unlimited. They combine dynamic programming and implicit enumeration of patterns: assuming a threshold value as a valid lower bound, they enumerate all feasible cutting patterns providing a value better than the threshold. If there are no such patterns, they correct the threshold value and run the method again. For the bounded version of the four-stage version of the problem, Clautiaux et al. [6] proposed a hypergraph based label setting algorithm. An initial dynamic program, represented by a hypergraph, is created to solve the unbounded problem. Then a filtering procedure based on Lagrangian relaxation is used to remove non-promising hyperarcs from the hypergraph. Finally, a label setting algorithm is applied to solve the 2KP by handling incrementally item upper bound constraints one by one. Note that when the number of cutting stages is limited to two or three, ILP models perform well as shown by Lodi and Monaci [16] and Puchinger and Raidl [22].

Leftovers complicate significantly the problem, mainly because the objective function has to account for the total length of the last cutting pattern. The rounded up lower bound obtained by column generation is seldom equal to the optimal solution value. An application of the cutting-stock problem with leftovers to glass cutting was treated by Puchinger et al. [23]. The problem they considered is to minimize the number of plates and the length of the used part of the last plate. Extra constraints are imposed on the order in which items are cut due to storage considerations. The authors have designed heuristics, meta-heuristics and heuristic branch-and-bound methods to solve the problem. Recently, similar approaches have been proposed by Dusberger and Raidl [8]. They developed a Variable Neighbourhood Search based on a Ruin-and-Recreate principle. A cutting pattern for a plate is represented by a cutting tree. Heuristics aim to remove some parts of the cutting tree for some plates (ruin step), then a new solution is obtained from items removed by the ruin step (recreate step). Recreate step is based on a constructive heuristic; the authors provide an ILP model to handle this part. Dusberger and Raidl [9] then extended their own work by adding dynamic programming in their recreate step. Instead of using heuristics, Andrade et al. [2] proposed direct MILP models for the 2BP with leftover inspired from the works of Lodi and Monaci [16] and with variables and constraints to handle leftover.

We are not aware of existing works in the literature studying our variant

of the cutting-stock problem with four stage cutting, possible rotation, and demands. Moreover, we want to handle leftovers and solve real-life size instances with large plates (typically 6000×3000) and many items (typically more than 100).

3. Extended formulations for 2DGCSPL

A classical approach solving for hard packing problems is to use extended formulations resulting from a Dantzig-Wolfe decomposition. In this section, we first describe how this decomposition can be applied to the 2DGCSPL, yielding an exponential size reformulation. The latter is to be solved using column generation and we explain how the pricing subproblem can be solved. Finally, we describe a pseudo-polynomial size extended formulation.

Recall that for 2DGCSPL, one needs to pack set $\mathcal{I}_b$ of items to three type of plates: a leftover plate from batch $b-1$ of given dimension $W' \times H$ (type 1), an unlimited number of standard plates of dimension $W \times H$ (type 2), and a single potential leftover plate of dimension $W \times H$ (type 3); and only a part of the width of the plate of type 3 is used (it is equals to the width of the cutting pattern assigned to it). Using these three types, the objective is to minimize the total used plate width and assuring that exactly one plate of type 1 and at most one plate of type 3 are used respectively. For the first batch, the plate of type 1 does not exist; the model can easily be adapted to this special case.

3.1. An extended formulation obtained by Dantzig-Wolfe decomposition

To formulate 2DGCSPL, let $\mathcal{P}_t$ be the set of all valid cutting patterns for a plate of type $t \in \{1,2,3\}$. Let a_i^p and w^p be the number of items $i \in \mathcal{I}$ in cutting pattern $p \in \mathcal{P}_t$ and its width. Let also λ_p be an integer variable which is equal to the number of times pattern $p \in \mathcal{P}_t$ is used in the solution. Then the 2DGCSPL can be formulated as

$$\min \sum_{p\in\mathcal{P}_1} W'\lambda_p + \sum_{p\in\mathcal{P}_2} W\lambda_p + \sum_{p\in\mathcal{P}_3} w^p\lambda_p \tag{1}$$

$$\sum_{p\in\mathcal{P}_1\cup\mathcal{P}_2\cup\mathcal{P}_3} a_i^p\lambda_p = d_i, \qquad \forall i \in \mathcal{I}, \tag{2}$$

$$\sum_{p\in\mathcal{P}_t} \lambda_p = 1, \qquad \forall t \in \{1,3\}, \tag{3}$$

$$\lambda_p \in \mathbb{Z}_+, \qquad \forall p \in \mathcal{P}_2, \tag{4}$$

$$\lambda_p \in \{0,1\}, \qquad \forall p \in \mathcal{P}_t, t \in \{1,3\}. \tag{5}$$

Objective function (1) minimizes the total used plate width. Note that the first term is a constant, since the first plate has to be used. Constraints (2) guarantee that the item demands are satisfied. Constraint (3) forces one to use exactly one plate of the first and the third type (the latter can have the width equal to zero, so implicitly we enforce an upper bound of one on

type 3 plates). The remaining constraints are added to ensure integrality of the variables. Note that the number of variables λ is exponential. A standard way to tackle formulation (1)–(5) is to solve its linear relaxation, called Master Problem (MP) using a column generation procedure: one alternates between solving the Restricted Master Problem (RMP), which is the MP with a restricted number of variables λ, and the pricing problem. The latter searches for negative reduced cost variables λ to be added to the RMP. Consider a dual solution (π, μ) of the RMP, where vector π is associated to constraints (2) and vector $\mu = (\mu_1, \mu_3)$ is associated to constraints (3). The pricing problem is to find a cutting pattern $p \in \mathcal{P}_1 \cup \mathcal{P}_2 \cup \mathcal{P}_3$ that minimizes the following reduced cost

$$\bar{c}_p = -\sum_{i \in \mathcal{I}} a_i^p \pi_i + \begin{cases} W' - \mu_1, & p \in \mathcal{P}_1, \\ W, & p \in \mathcal{P}_2, \\ w^p - \mu_3, & p \in \mathcal{P}_3. \end{cases} \tag{6}$$

Obviously, the pricing problem decomposes into three subproblems, one for each plate type.

3.2. Solving the column-generation pricing problem

Each pricing subproblem is a bounded four-stage restricted guillotine-cut two-dimensional knapsack problem. A dynamic programming algorithm (DP) for the unbounded case of this problem (based on the one of Beasley [3]) was presented by us in Clautiaux et al. [6]. To be self-contained, we recall the dynamic programming recursion here. Then, we restate the complete formulation of the bounded case as an ILP model.

When item rotations are allowed, set $\mathcal{I}$ of items is duplicated in the subproblem to include rotated item copies. The two copies of an item i share the same dual value π_i from the RMP, and the same upper bound d_i.

Since we use restricted cuts, we consider two types of dynamic programming states: those related to a restricted cut (used to initiate a strip), and those that are used to complete a strip.

Let (w, h, s), $w \in \{1, \ldots, W\}$, $h \in \{1, \ldots, H\}$, $s \in \{1, \ldots, 3\}$, be the state corresponding to the situation in which a rectangular part of dimension $w \times h$ of the plate is to be separated from the current plate using a guillotine cut of stage s. Let $(\overline{w, h, s})$, $w \in \{1, \ldots, W\}$, $h \in \{1, \ldots, H\}$, $s \in \{2, \ldots, 4\}$, be the state corresponding to the same situation with the additional restriction that the next cut should obtain a single item copy. For a given state (w, h, s), let $U(w, h, s)$ be the maximum value of a configuration obtained from this state. Note that this value only depends on the dual values π associated to item copies, and an additional term for type 3 plate only, which is equal to the total width used. It is sufficient to account for this restricted width in stage 1 cuts only. Then $U(W, H, 1)$ equals to the pricing subproblem optimum value (ignoring constant values in (6)).

Let $\mathcal{W}(w, h)$ and $\mathcal{H}(w, h)$ be the set of all possible widths and heights of items which fit into rectangle $w \times h$.

$$\mathcal{W}(w, h) = \bigcup_{i \in \mathcal{I}:\ w_i \leqslant w, h_i \leqslant h} \{w_i\}, \quad \mathcal{H}(w, h) = \bigcup_{i \in \mathcal{I}:\ w_i \leqslant w, h_i \leqslant h} \{h_i\}.$$

We now give the recursion for computing values $U(w,h,s)$ and $U(\overline{w,h,s})$ for type 2 plates.

$$U(w,h,1) = \max\left\{0, \max_{w' \in \mathcal{W}(w,h)} \left\{U(\overline{w',h,2}) + U(w-w',h,1)\right\}\right\}$$
$$U(w,h,2) = \max\left\{0, \max_{h' \in \mathcal{H}(w,h)} \left\{U(\overline{w,h',3}) + U(w,h-h',2)\right\}\right\}$$
$$U(w,h,3) = \max\left\{0, \max_{w' \in \mathcal{W}(w,h)} \left\{U(\overline{w',h,4}) + U(w-w',h,3)\right\}\right\}$$

$$U(\overline{w,h,2}) = \max_{i \in \mathcal{I}:\ w_i = w, h_i \leqslant h} \{\pi_i + U(w,h-h_i,2)\}$$
$$U(\overline{w,h,3}) = \max_{i \in \mathcal{I}:\ h_i = h, w_i \leqslant w} \{\pi_i + U(w-w_i,h,3)\}$$
$$U(\overline{w,h,4}) = \max\left\{0, \max_{i \in \mathcal{I}:\ w_i = w, h_i \leqslant h} \left\{\pi_i + U(\overline{w,h-h_i,4})\right\}\right\}$$

In the recursive formulae for $U(w,h,s)$ and $U(\overline{w,h,4})$, the alternative with zero value corresponds to turning the remaining rectangular part of the plate into waste.

For the type 1 plate generation, the recursions are identical, only initialization needs to be adapted to the available width. For type 3 plates, the recursive formula for $U(w,h,1)$, becomes

$$U(w,h,1) = \max\left\{0, \max_{w' \in \mathcal{W}(w,h)} \left\{U(\overline{w',h,2}) - w' + U(w-w',h,1)\right\}\right\}$$

where term $-w'$ represents the penalty for the width consumed.

Using the formalism of Martin et al. [19], our dynamic program may be reformulated as a max cost flow problem in a directed acyclic hypergraph. In Clautiaux et al. [6], this transformation as well as pre-processing procedures for this hypergraph are presented. Let $G = (\mathcal{V}, \mathcal{A})$ denote this directed acyclic hypergraph. Vertex set $\mathcal{V}$ is composed of all states from the dynamic program and also of special states related to single items or waste. Set $\mathcal{S}$ of these special states are sources of the hypergraph. Its sink u is the vertex associated to state $(W,H,1)$. Each hyperarc a has a head $h(a)$, which contains a unique vertex, and a tail multiset $\mathcal{T}(a)$, which contains one or more vertices representing subplate cutting patterns, each of which can occur more than once. A hyperarc a represents the cutting decision that turns state $h(a)$ into states in $\mathcal{T}(a)$. A vertex v has a hyperarc predecessor (resp. successor) set $\Gamma^-(v)$ (resp. $\Gamma^+(v)$).

Let $\mathcal{A}(i)$ be the set of hyperarcs whose tail sets include the source node representing item $i \in \mathcal{I}$. Let x_a, for $a \in \mathcal{A}$, be the integer variable representing the flow value going through hyperarc a. The resulting flow model associated

to the pricing problem can therefore be written as an Integer Linear Program:

$$\max f_p = \sum_{i \in \mathcal{I}} \pi_i \sum_{a \in \mathcal{A}(i)} x_a \tag{7}$$

$$\text{s.t.} \sum_{a \in \Gamma^-(v)} x_a - \sum_{a' \in \Gamma^+(v)} x_{a'} = 0, \quad \forall v \in \mathcal{V} \backslash \{u \cup \mathcal{I} \cup \varnothing\} \tag{8}$$

$$\sum_{a \in \Gamma^-(u)} x_a = 1 \tag{9}$$

$$\sum_{a \in \mathcal{A}(i)} x_a \leqslant d_i, \quad \forall i \in \mathcal{I} \tag{10}$$

$$x_a \in \mathbb{N}, \quad \forall a \in \mathcal{A} \tag{11}$$

Objective function (7) aims to maximise the reduced cost of items. This objective yields the non-constant term of the reduced cost $f_p = \sum_i a_i^p \pi_i$ in (6). Constraints (8)-(9) are classical flow conservation constraints. They ensure that a valid pattern is built according to cutting properties. Constraint set (10) avoids overproduction of an item. When $p \in \mathcal{P}_3$, value w_p in (6) is computed as: $w_p = \sum_{a \in \bar{\mathcal{A}}} w_a x_a$, with $\bar{\mathcal{A}}$ the set of hyperarcs related to a cut of first stage. Objective function (7) therefore becomes $\max_{p \in P_3} \{f_p - w_p\}$.

Tackling the pricing problem by applying an ILP solver to formulation (7)–(11) leads to large computing times. We have shown in Clautiaux et al. [6] that an iterative method based on state-space relaxations allows one to solve the pricing problem significantly faster. However, even with the latter approach remains too slow to be used inside column generation. As explained in Section 5, our method will rather rely on solving the unbounded version of the pricing problem.

3.3. Deriving a pseudo-polynomial flow formulation for 2DGCSPL

Building on our pricing problem network flow reformulation, one can formulate our cutting-stock problem as an arc-flow model yielding an ILP formulation of pseudo-polynomial size (as done by Valério de Carvalho [27] for the 1BP).

Let $G_t = (\mathcal{V}_t, \mathcal{A}_t)$ be the hypergraph related to a plate of type $t \in \{1, 2, 3\}$. Let $z_t, t \in \{1, 2, 3\}$ be the total number of patterns of type t that are used (z_1 and z_3 are set to be equal to one, whereas z_2 is a variable). Then, the 2DGCSPL

model can take the form:

$$\min W' z_1 + W z_2 + z_3 \sum_{a \in \bar{\mathcal{A}}_3} w_a x_a \tag{12}$$

$$\text{s.t.} \sum_{a \in \Gamma^-(v)} x_a - \sum_{a' \in \Gamma^+(v)} x_{a'} = 0, \quad \forall v \in \mathcal{V}_t \backslash \{u_t \cup \mathcal{I} \cup \varnothing\}, \forall t \in \{1,2,3\}, \tag{13}$$

$$\sum_{a \in \Gamma^-(u_t)} x_a = z_t, \quad \forall t \in \{1,2,3\} \tag{14}$$

$$\sum_{t \in \{1,2,3\}} \sum_{a \in \mathcal{A}_t(i)} x_a = d_i, \quad \forall i \in \mathcal{I} \tag{15}$$

$$z_1 = z_3 = 1, \quad z_2 \in \mathbb{N} \tag{16}$$

$$x_a \in \mathbb{N}, \quad \forall a \in \mathcal{A}_t, \forall t \in \{1,2,3\} \tag{17}$$

Objective (12) is equivalent to (1). Constraints (13)-(14) are flow conservation constraints among the different plate types. Constraints (15) ensure that the demand is fulfilled. Variable definitions (16) ensure to select only one pattern from types 1 and 3. Finally, each hyperarc variable is integer (17).

4. Constructive and evolutionary heuristics

Due to the large size of instances we aim to solve, a natural approach is to use heuristics, which are usualy used in the literature. In this section, we first propose heuristics to solve the bounded case of the pricing problem. Such a choice is motivated by the fact that solving this problem to optimality is computationally expensive in the column generation context. Second, we also derive constructive heuristics for 2DGCSPL. They will be used in computational comparisons with diving heuristics presented below.

We designed two heuristics for the pricing problem. The first is based on a hypergraph exploration. The second is an evolutionary algorithm.

4.1. Hypergraph based heuristics for the pricing problem with bounded production

As described in Section 3.2, the dynamic program seeks a max-cost flow to the sink in the acyclic hypergraph, where vertices correspond to sub-plate cutting patterns and hyperarcs correspond to combinations of these patterns. Thus a cutting pattern can be represented by a set of hyperarcs (how sub-plates were combined) and a set of vertices (intermediate states). As a first step of the heuristic, we run the dynamic program for the unbounded problem. It allows us to compute the best reduced cost $\bar{c}_v$ associated with each vertex $v \in \mathcal{V}$, *i.e.* the dynamic programming value associated to the corresponding state. The reduced cost associated with an hyperarc $\bar{c}_a$ is the sum of the reduced costs of its tails : $\bar{c}_a = \sum_{v \in \mathcal{T}(a)} \bar{c}_v$. We also define as $y(\mathcal{A}') \in \mathbb{Z}_+^{|\mathcal{I}|}$ the partial solution corresponding to the source vertices in the tail sets of hyperarcs in multiset $\mathcal{A}'$: $y_i(\mathcal{A}') = \mid \mathcal{A}' \cap \mathcal{A}(i) \mid, \forall i \in \mathcal{I}$.

The constructive stage of the first heuristic is run for a given hyperarc $a' \in \mathcal{A}$ and a given partial solution $y' \in \mathbb{Z}_+^{|\mathcal{I}|}$, $y' \leqslant d$. We start by adding $y(\{a'\})$ to y'. At any time, we keep the current set $\mathcal{L}$ of *open vertices*, initialized with all non-source vertices in the tail set of hyperarc a'. In every iteration of the algorithm, a vertex v' is selected in $\mathcal{L}$ and removed from it. Then, a hyperarc a incoming to v' with the smallest reduced cost such that $y' + y(\{a\}) \leqslant d$ is chosen, all non-source vertices in the tail set of a are added to $\mathcal{L}$, and we pass to the next iteration. The algorithm stops when set $\mathcal{L}$ becomes empty. It returns the obtained solution y' and the corresponding multiset of hyperarcs $\mathcal{A}'$. The pseudocode of this heuristic is presented as Function HG-Constr-Heur.

Function HG-Constr-Heur(a', y')

$\mathcal{L} \leftarrow \emptyset$; $\mathcal{A}' \leftarrow \emptyset$;
if $y' + y(\{a'\}) \leqslant d$ **then**
  $y' \leftarrow y' + y(\{a'\})$; $\mathcal{A}' \leftarrow \{a'\}$; $\mathcal{L} \leftarrow \mathcal{T}(a') \backslash \mathcal{S}$;
while $\mathcal{L} \neq \emptyset$ **do**
  Pick $v \in \mathcal{L}$; $\mathcal{L} \leftarrow \mathcal{L} \backslash \{v\}$;
  $\mathcal{A}'' \leftarrow \{a \in \Gamma^-(v) : \ y' + y(\{a\}) \leqslant d\}$;
  if $\mathcal{A}'' \neq \emptyset$ **then**
    $a'' \leftarrow \operatorname{argmax}_{a \in \mathcal{A}''} \{\bar{c}_a\}$;
    $y' \leftarrow y' + y(\{a''\})$; $\mathcal{A}' \leftarrow \mathcal{A}' \cup \{a''\}$; $\mathcal{L} \leftarrow \mathcal{L} \cup \mathcal{T}(a'') \backslash \mathcal{S}$;
return $(y', \mathcal{A}')$

The constructive heuristic can then be embedded in a local search method. First the constructive heuristic finds the best feasible solution among all hyperarc $a' \in \Gamma^-(u)$ incoming to the sink node u, and best solution $(y^*, \mathcal{A}^*)$ is obtained. Then the following local search algorithm is applied to this solution. At every iteration, each hyperarc $\hat{a} \in \mathcal{A}'$ is replaced by another hyperarc $\tilde{a} \in \Gamma^-(\mathcal{H}(\hat{a})), \tilde{a} \neq \hat{a}$ incoming to the same vertex as $\hat{a}$, and solution y' is modified accordingly. If an improved solution is found, next iteration starts. The process stops when no improvement occurred or the iteration number limit is reached (minimum between $|\mathcal{I}|$ and 50). The formal presentation of this heuristic is given as Algorithm 1.

4.2. Evolutionary heuristic for the pricing problem with bounded production

The second heuristic is an evolutionary algorithm inspired from Hadjiconstantinou and Iori [14]. Let $\bar{\mathcal{I}}$ be an unary representation of item set $i \in \mathcal{I}$, *i.e.* d_i copies of each item i are created. We use an indirect encoding: each individual (or genome) g is represented by a sequence of integers $g_1, \ldots, g_{n(g)}$, each of them refers to the g_j-th item to cut in $\bar{\mathcal{I}}$. Size $n(g)$ of a genome g is usually smaller than $|\bar{\mathcal{I}}|$, as, generally, not all item copies fit into a single plate. To obtain the solution and its value from a genome, we run the following so-called *First-Fit* heuristic.

Algorithm 1: Hypergraph based heuristic

$(y^*, \mathcal{A}^*) = (0, \varnothing)$;
for $a' \in \Gamma^-(u)$ **do**
  $(y, \mathcal{A}) \leftarrow$ HG-Constr-Heur$(a', 0)$;
  if $\pi y > \pi y^*$ **then** $(y^*, \mathcal{A}^*) = (y, \mathcal{A})$;
$k \leftarrow 0$;
repeat
  $improve \leftarrow false$; $k \leftarrow k + 1$;
  $(y', \mathcal{A}') = (y^*, \mathcal{A}^*)$;
  for $\hat{a} \in \mathcal{A}'$ **do**
    Let $\hat{\mathcal{A}}$ be the part of solution (flow) $\mathcal{A}'$ coming to $\mathcal{H}(\hat{a})$;
    for $\tilde{a} \in \Gamma^-(\mathcal{H}(\hat{a})), \tilde{a} \neq \hat{a}$ **do**
      $(\tilde{y}, \tilde{\mathcal{A}}) \leftarrow$ HG-Constr-Heur$\big(\tilde{a}, y' - y(\hat{\mathcal{A}})\big)$;
      if $\pi \cdot (y' - y(\hat{\mathcal{A}}) + \tilde{y}) > \pi y^*$ **then**
        $y^* \leftarrow y' - y(\hat{\mathcal{A}}) + \tilde{y}$; $\mathcal{A}^* \leftarrow \mathcal{A}' \backslash \hat{\mathcal{A}} \cup \tilde{\mathcal{A}}$;
        $improve \leftarrow true$;
until $\neg improve$ **or** $k = \min\{|\mathcal{I}|, 50\}$;
Store solution $(y^*, \mathcal{A}^*)$;

First-Fit heuristic cuts items in the order given by genome g. Let $\mathcal{R}$ be the stack of available sub-plates initialized with initial plate $(W, H, 1)$ at stage 1. First-Fit heuristic takes the first available plate in $\mathcal{R}$, removes it from $\mathcal{R}$, and then tries to cut the first available item l_j in the order given by the genome. Since guillotine cuts are considered here, this implies that this cut always divides the current sub-plate r into two smaller sub-plates r' and r'' which are added to stack $\mathcal{R}$. Note that if no remaining item in the genome fits in a given plate r, we use other items which fit into it if possible, thus increasing the solution quality. If no items fit into a sub-plate, it is discarded. The process stops when $\mathcal{R} = \varnothing$.

The purpose of the evolutionary algorithm is to create a new population containing improving solutions from the previous one. We initialize our evolutionary algorithm with a population of $|\mathcal{I}|$ individuals corresponding to different random permutations of multiset $\bar{\mathcal{I}}$. To ensure good quality solutions, the initial population is then reduced using an elitist strategy. This one aims to keep the pool F of p_{size} best individuals from the initial population. Starting from this pool, a new population is created in two phases: new individual generation and crossover operation. The first phase is the same as the initial population generation. This enriches the pool by p_{size} new individuals. In the second phase, the offspring set O of individuals is produced using a two-point crossover where only 25% of offspring individuals are randomly kept.

Two-point crossover uses as input two parent individuals represented by their genomes g_1 and g_2 and create a new one g_3 by selecting parts of parent genomes.

Two positions in the genome of the first parent p_1 and p_2 are randomly picked. The offspring is then created by merging genome of parent g_1 between positions 0 and p_1, genome of parent g_2 between positions p_1 and p_2 and then genome of parent g_1 between positions p_2 and $n(g_1)$.

After a crossover phase, all remaining individuals solutions are moved in pool F. Then only p_{size} best individuals in F are kept for the next population generation. To have enough solutions from one population generation to another, the population size p_{size} is set to 20. The total number of population generations is set to $|\mathcal{I}|/2$.

4.3. Heuristics for the 2DGCSPL

A direct way to obtain a heuristic solution for the 2DGCSPL is to use the evolutionary algorithm described in Section 4.2 iteratively for each plate. We call this heuristic (ea).

Alternatively one can use standard bin-packing list heuristics such as *Next-Fit* (NF), *Best-Fit* (BF), *First-Fit* (FF) or *Bottom-Left* (BL). These heuristics run in polynomial time and produce a feasible packing for our problem. The main difference between them is the packing strategy for a given item. Assume a set of items to cut $\mathcal{I}$ and a list of empty sub-plates $\mathcal{R}$. In Next-Fit heuristic, a given item $i \in \mathcal{I}$ is packed in the next sub-plate $r \in \mathcal{R}$ in which it fits. If there are previous sub-plates in which the item does not fit, they are simply removed from $\mathcal{R}$. In First-Fit heuristic, a given item $i \in \mathcal{I}$ is packed in the first sub-plate $r \in \mathcal{R}$ in which it fits. In Best-Fit heuristic, a given item $i \in \mathcal{I}$ is packed in the best sub-plate $r \in \mathcal{R}$ in which it fits. The best sub-plate is the one such that after packing an item its remaining horizontal or vertical unused space is minimised. In Bottom-Left heuristic, a given item $i \in \mathcal{I}$ is packed in the sub-plate $r \in \mathcal{R}$ in which it fits such that position of the item is the closest to the bottom left corner of the bin. At any step of these heuristics, if no sub-plates can accommodate the item, a new one is created, which triggers the addition of a new bin. More details about these heuristics can be found in Lodi et al. [15].

We now present heuristic algorithm (iub) which is more time consuming than (ea). It combines several list heuristics as well the evolutionary algorithm. We consider $10 * |\mathcal{I}|$ random permutations of set $\bar{\mathcal{I}}$. Four list heuristics mentioned above are applied for each of these permutations. These heuristics can be used in several ways. The first option is to fill one plate at a time, *i.e.* we switch to the next plate only when there are no items in the current list that fit into the current plate. The second option is to open all bins at the same time.

The third option is to implement the following two-phase process. A plate of infinite width is created and items are packed in it using the list heuristics above. According to the guillotine cut property, the obtained solution can be represented by a certain amount of vertical strips. We collect the obtained strips and assign them to original plates using BF, FF and NF list heuristics as in 1BP.

At the same time, we also generate an extra bin-packing solution by solving iteratively 2KPs with our evolutionary algorithm. For the latter we generate 10 times an initial random population and improve it during $|\mathcal{I}|/10$ iterations.

Using the three described options as well as the evolutionary algorithm, we obtain a set of feasible solutions for each random permutation of set $\bar{\mathcal{I}}$. We first record the best solution providing us a valid upper bound for our cutting problem. Then, among all cutting-stock solutions we select a plate of smallest waste, and we add it to the partial solution. We then reiterate the whole process on the residual problem. The algorithm terminates when there are no more items to cut.

5. A diving heuristic for the 2DGCSPL

In this section we present our diving heuristic to solve 2DGCSPL. We first introduce standard column generation based diving algorithms, following Sadykov et al. [25]. Then we show how these algorithms can be adapted when production bound constraints are not enforced in the pricing subproblem. Several additional ingredients are presented to improve the quality of the solutions obtained.

5.1. Diving heuristics

A pure column generation based diving heuristic, as presented in Sadykov et al. [25], is a depth-first search heuristic in a branch-and-price enumeration tree obtained when branching on the master problem variables λ. It can be directly applied to our problem in the following way. At each iteration one (re)optimizes the master linear problem; as a result, one obtains a fractional solution $\bar{\lambda}$. One chooses a component λ_p of the solution vector such that its current value $\bar{\lambda}_p$ is the closest to its nearest non-zero integer $\lceil \bar{\lambda}_p \rfloor$. Then one sets $\lambda_p = \lceil \bar{\lambda}_p \rfloor$ in the partial solution, and updates the demand of items: $d \leftarrow d - a^p \lceil \bar{\lambda}_p \rfloor$. The algorithm proceeds to the next iteration with the updated (or residual) master problem. The process terminates when the partial solution becomes complete (*i.e.* when all master constraints are satisfied). Note that Furini et al. [11] used this pure diving heuristic for two-stage 2D guillotine cutting-stock problem.

As shown in Sadykov et al. [25], effectiveness of the diving heuristic significantly increases when it is coupled with Limited Discrepancy Search. This variant allows one to do a limited number of backtracks in the search tree. During a backtrack, the decision to add a column to the partial solution is reverted. Once this happens, this column is added to the tabu list. Columns in the tabu list cannot be added to the partial solution.

To improve the convergence of the column generation procedure on which the diving heuristic relies, we use the automatic dual price smoothing stabilization proposed in Pessoa et al. [20].

5.2. Diving with non proper columns

To achieve primal feasibility, the basic diving heuristic requires to use so-called *proper* columns, *i.e.* variables that could take a non-zero value in an integer solution of the residual master problem. In our context, a variable λ_p, $p \in \mathcal{P}_1 \cup \mathcal{P}_2 \cup \mathcal{P}_3$, is proper if $a_i^p \leqslant d_i, \forall i \in \mathcal{I}$. Therefore, in each pricing subproblem we should impose upper bounds on the number of copies of items

in the cutting pattern. However when solving the pricing problem (6) in Section 3.2, we consider only the unbounded version. In the presence of these upper bounds, the pricing problem becomes significantly harder to solve to optimality. A possible solution to this issue is to solve the pricing problem only heuristically using algorithms from Sections 4.1 and 4.2. However, our preliminary experiments showed that this approach deteriorates significantly the quality of solutions obtained by the diving heuristic.

In this work, we propose a variant of the diving heuristic which uses non-proper columns. A previous study of Cintra et al. [5] and our preliminary experiments showed that the lower bound obtained by solving the MP with non-proper columns is close to the one obtained when using exclusively proper columns. As the quality of diving heuristics depend mainly on the strength of the MP bound, we may then expect that a "non-proper" diving heuristic will be efficient for our problem.

Our "non-proper" diving heuristic proceeds as follows. Remember that at each iteration, the residual master problem is solved by column generation. Both proper and non-proper columns may be generated. However, the partial solution can only be augmented with proper columns. Therefore, given a fractional solution $\bar{\lambda}$, we choose a *proper* variable λ_p with a value closest to its nearest non-zero integer. If such variable exists, we proceed the same way as in the basic diving heuristic. If there is no such "*proper*" variable, we choose a column λ_p with the smallest reduced cost (with respect to the optimal dual solution of the master problem) among

- all proper columns contained in the current RMP;
- and proper columns generated by solving the bounded pricing problem by heuristic algorithms presented in Sections 4.1 and 4.2.

As usual, this column is then added to the partial solution with the value equal to the nearest non-zero integer of $\bar{\lambda}_p$. In particular, if $\bar{\lambda}_p = 0$ then $\lambda_p = 1$ is included in the partial solution.

Our preliminary experiments showed that fixing a type 3 columns λ_p, for $p \in \mathcal{P}_3$, early in the search has a negative impact on the quality of solutions obtained. Therefore, we adopt the following modification. Cutting patterns $p \in \mathcal{P}_3$ are never added to the partial solution before patterns $p \in \mathcal{P}_1 \cup \mathcal{P}_2$. As there is exactly one pattern $p \in \mathcal{P}_3$ in any feasible solution, once it is added to a partial solution, the latter should become complete. Therefore, each time the partial solution is augmented with a cutting pattern of type 1 or 2, we verify heuristically whether the remaining item copies can be cut into one plate of dimension $W \times H$. If it is possible, we produce a cutting pattern $p \in \mathcal{P}_3$ including all remaining item copies and minimizing heuristically its width w^p. Then this pattern is used to complete the solution, and the diving heuristic terminates.

One can develop further this idea and, each time the partial solution is augmented, to formulate the residual 2DGCSPL and solve it with heuristics described in Sections 4.1 and 4.2. Calls to these heuristics are done iteratively

for each plate until the residual problem instance is closed. This modification can be seen as a combination of the diving and pricing heuristics. This combined heuristic is presented in Algorithm 2. A boolean parameter *lastPlateOnly* is used to set how to evaluate the residual 2DGCSPL instance: building only a solution for the last plate or a complete solution to the residual 2DGCSPL instance.

Algorithm 2: The combined pricing heuristics and diving heuristic with non-proper columns

$\mathcal{P}^* \leftarrow$ solution of the 2DGCSPL with demands d by the evolutionary algorithm
$d' \leftarrow d$, $\mathcal{P}^{part} \leftarrow \varnothing$
repeat
  Solve the MP with production bounds d' by column generation and obtain solution $\bar{\lambda}$
  $\mathcal{P}^{prop} \leftarrow \{p \in \mathcal{P}_1 \cup \mathcal{P}_2 : \bar{\lambda}_p > 0, a^p \leqslant d'\}$ (set of proper patterns in the solution)
  if $\mathcal{P}^{prop} \neq \varnothing$ **then**
    $p' \leftarrow \text{argmin}_{p \in \mathcal{P}^{prop}} \{| \bar{\lambda}_p - \lceil \bar{\lambda}_p \rfloor |\}$
  else
    $\mathcal{P}^{\text{RMP}} \leftarrow$ set of patterns of type 1 and 2 in the RMP
    $\mathcal{P}^{heur} \leftarrow$ set of heuristic solutions to the pricing problem of type 1 and 2
    $p' \leftarrow \text{argmin}_{p \in \mathcal{P}^{\text{RMP}} \cup \mathcal{P}^{heur}} \{\bar{c}_p\}$
  $\mathcal{P}^{part} \leftarrow \mathcal{P}^{part} \cup \{p'\}$, $d' \leftarrow d' - a^{p'}$
  if $lastPlateOnly = true$ **then**
    $p^3 \leftarrow$ heuristic solution to the pricing problem of type 3
    if $d' - a^{p^3} = \mathbf{0}$ **then**
      $d' \leftarrow \mathbf{0}$
      if $cost\big(\mathcal{P}^{part} \cup \{p^3\}\big) < cost(\mathcal{P}^*)$ **then** $\mathcal{P}^* \leftarrow \mathcal{P}^{part} \cup \{p^3\}$
  else
    $\mathcal{P}^{ev} \leftarrow$ heuristic solution to the residual 2DGCSPL with demand d'
    if $cost\big(\mathcal{P}^{part} \cup \mathcal{P}^{ev}\big) < cost(\mathcal{P}^*)$ **then** $\mathcal{P}^* \leftarrow \mathcal{P}^{part} \cup \mathcal{P}^{ev}$
until $d' = 0$
return $\mathcal{P}^*$

To introduce a Limited Discrepancy Search (LDS) in our "non-proper" diving heuristic, we need to ensure that at least one non-tabu column is produced by our pricing problem heuristic. This can be achieved by generating a sufficient number of different cutting patterns, *i.e.* the size of set $\mathcal{P}^{heur}$ in Algorithm 2 is strictly larger that the current size of the tabu list.

Note that in addition to accelerating column generation, stabilization has an another positive effect. When it is used, the MP solution is generally more fractional and thus contains a larger number of columns. Therefore, there is a larger probability to have a proper column in the fractional solution.

5.3. Solving the pricing problem with partial enumeration

Although we have explained above that our diving heuristic can be adapted to handle non-proper columns, still the quality of the solutions may be decreased in comparison with the "proper" case. In this section, we propose a modification to our dynamic program that partially takes into account upper bounds on the number of item copies in order to favour the generation of proper cutting patterns. As it will be seen from our computational results, this approach allows one to improve the quality of the lower bound obtained by solving the MP as well as the quality of solutions produced by the diving heuristic. This comes at the cost of a slightly larger dynamic program. Nevertheless we show below that this can be controlled by a suitable configuration.

On one hand, complete enumeration of the possible patterns would take into account the bound constraints, but the computational cost would be huge. On the other hand, the dynamic program has a reasonable computational cost, but it does not take into account the bound constraints. Our idea is to mix both approaches, by replacing some parts of DP by a partial enumeration.

This idea is implemented using so-called *meta-items*, each one representing a partial vertical or horizontal stack of item copies satisfying production upper bounds. When restricted states $U(\overline{w,h,k}), k = 2,3,4$ are considered, instead of choosing one item to initiate the stripe, we choose a meta-item (or equivalently the items that it represents). There is potentially an exponential number of possible meta-items to initiate the stripe. Therefore, we introduce an additional parameter δ, which restricts the possible meta-items produced by only considering items whose width/height is close enough to the size of the stripe.

Formally, let $\bar{a}_i^m$ be the number of copies of item $i \in \mathcal{I}$ included into meta-item m. Given three values $0 < w \leqslant W$, $0 < h \leqslant H$, and $0 < \delta \leqslant \min_{i\in\mathcal{I}} w_i$, we define the following set $\mathcal{M}^x(w,h,\delta)$ of vertical meta-items. Each meta-item $m \in \mathcal{M}^x(w,h,\delta)$ forms in the cutting pattern a partial vertical stack of width w containing copies of items $i \in \mathcal{I}$ such that $w - \delta < w_i \leqslant w$ and $h_i \leqslant h$. Items that do not belong to m may only be cut in other vertical stacks or in the same stack above the item copies in m. In addition, copies of items $i \in \mathcal{I}$ such that $\bar{a}_i^m > 0$ may only be cut in other vertical stacks. Formally:

$$m \in \mathcal{M}^x(w,h,\delta) \Leftrightarrow \begin{cases} \exists i \in \mathcal{I}, w_i = w : \ \bar{a}_i^m > 0, \\ \bar{a}_i^m > 0 \Rightarrow w - \delta < w_i \leqslant w \text{ and } h_i \leqslant h, \quad \forall i \in \mathcal{I}, \\ \bar{a}_i^m \leqslant d_i, \quad \forall i \in \mathcal{I}, \\ \sum_{i\in\mathcal{I}} \bar{a}_i^m h_i \leqslant H. \end{cases}$$

The first condition ensures that one item has width w (and thus a restricted pattern is built). The second condition ensures that the size of the items in the meta-item satisfies the requested limitations. The third condition ensures that

the meta-item satisfies the production bound constraints. The fourth condition ensures that the meta-item height does not exceed the plate height.

Analogously, given values $0 < w \leqslant W$, $0 < h \leqslant H$, and $0 < \delta \leqslant \min_{i\in\mathcal{I}} h_i$, we define the following set $\mathcal{M}^y(h, w, \delta)$ of horizontal meta-items:

$$m \in \mathcal{M}^y(h,w,\delta) \Leftrightarrow \begin{cases} \exists i \in \mathcal{I}, h_i = h : \ \bar{a}_i^m > 0, \\ \bar{a}_i^m > 0 \Rightarrow h - \delta < h_i \leqslant h \text{ and } w_i \leqslant w, \quad \forall i \in \mathcal{I}, \\ \bar{a}_i^m \leqslant d_i, \quad \forall i \in \mathcal{I}, \\ \sum_{i\in\mathcal{I}} \bar{a}_i^m w_i \leqslant W. \end{cases}$$

For each meta-item $m \in \mathcal{M}$, we define its total value $\bar{\pi}_m = \sum_{i\in\mathcal{I}} \pi_i \bar{a}_i^m$. For each vertical meta-item $m \in \mathcal{M}^x$, we define its total height $\bar{h}_m = \sum_{i\in\mathcal{I}} \bar{a}_i^m h_i$, and for each horizontal meta-item $m \in \mathcal{M}^y$, we define its total width $\bar{w}_m = \sum_{i\in\mathcal{I}} \bar{a}_i^m w_i$.

Note that, by definition, $\mathcal{M}^x(w, h, \delta) = \varnothing$ if $w \notin \mathcal{W}(W, H)$, and $\mathcal{M}^y(h, w, \delta) = \varnothing$ if $h \notin \mathcal{H}(W, H)$. Suppose now that for each $w \in \mathcal{W}(W, H)$ a value δ_w, $0 \leqslant \delta_w \leqslant \min_{i\in\mathcal{I}} w_i$, is fixed, and for each $h \in \mathcal{H}(W, H)$ a value δ_h, $0 \leqslant \delta_h \leqslant \min_{i\in\mathcal{I}} h_i$, is fixed. Then the recursive formulae for states $(\overline{w, h, s})$ can be rewritten in the following way without loss of any proper patterns from the set of feasible solutions.

$$U(\overline{w,h,2}) = \begin{cases} \max\limits_{m\in\mathcal{M}^x(w,h,\delta_w):\ \bar{h}_m\leqslant h} \left\{\bar{\pi}_m + U(w, h - \bar{h}_m, 2)\right\}, & \text{if } \delta_w > 0, \\ \max\limits_{i\in\mathcal{I}:\ w_i=w, h_i\leqslant h} \left\{\pi_i + U(w, h - h_i, 2)\right\}, & \text{if } \delta_w = 0, \end{cases}$$

$$U(\overline{w,h,3}) = \begin{cases} \max\limits_{m\in\mathcal{M}^y(h,w-\delta_w,\delta_h):\ \bar{w}_m\leqslant w} \left\{\bar{\pi}_m + U(w - \bar{w}_m, h, 3)\right\}, & \text{if } \delta_h > 0, \\ \max\limits_{i\in\mathcal{I}:\ h_i=h, w_i\leqslant w-\delta_w} \left\{\pi_i + U(w - w_i, h, 3)\right\}, & \text{if } \delta_h = 0, \end{cases}$$

$$U(\overline{w,h,4}) = \begin{cases} \max\left\{0, \max\limits_{m\in\mathcal{M}^x(w,h-\delta_h,1)} \{\bar{\pi}_m\}\right\}, & \text{if } \delta_w > 0, \\ \max\left\{0, \max\limits_{i\in\mathcal{I}:\ w_i=w, h_i\leqslant h-\delta_h} \left\{\pi_i + U(\overline{w, h - h_i, 4})\right\}\right\} & \text{if } \delta_w = 0, \end{cases}$$

If all values δ are fixed to zero, the modified dynamic program reduces to the original one presented in Section 3.2. The larger the values δ are, the fewer non-proper cutting patterns are generated and, at the same time, the larger is the number of meta-items. So, there is a trade-off between the complexity (or the size) of the dynamic program and the strength of the approximation of the space of proper cutting patterns by the space of feasible solutions of the dynamic program. We parametrize this trade-off by defining thresholds $\Delta^{size} \geqslant 0$ on the size of the sets of meta-items and $\Delta^{\text{diff}} \geqslant 0$ on values δ, respectively for dimension w and h. Given these thresholds, values δ are determined the following way. For each $w \in \mathcal{W}(W, H)$, we set δ_w to the largest value $\delta \leqslant \min\left\{\Delta_w^{\text{diff}}, \min_{i\in\mathcal{I}} w_i\right\}$ such that $\mid \mathcal{M}^x(w, H, \delta) \mid \leqslant \Delta_w^{size}$. As $\mathcal{W}^x(w, H, 0) = \varnothing$ for any w, such value δ always exists. Analogously, for each $h \in \mathcal{H}(W, H)$, we set δ_h to the largest value $\delta \leqslant \min\left\{\Delta_h^{\text{diff}}, \min_{i\in\mathcal{I}} h_i\right\}$ such that $\mid \mathcal{M}^y(h, W, \delta) \mid \leqslant \Delta_h^{size}$. Note again that $\mathcal{W}^y(h, W, 0) = \varnothing$ for any h. The sets of meta-items are computed by enumeration.

To illustrate such generation of meta-items, let us consider an instance with a raw plate of size $(4, 3)$ and three items $1 \times i_1 = (4, 1), 2 \times i_2 = (3, 1), 1 \times i_3 = (2, 1)$. We compute set $\mathcal{M}^x(4, 3, \delta)$ with $\delta = 2$. Initially meta-items composed of only items i_1 are created (see Figure 2-(b)). Secondly extra items are added to meta-items derived from item i_1. Since $\delta = 2$, only item i_2 is added (see Figure 2-(c) and Figure 2-(d)). From each created meta-item, a valid cutting pattern can be obtained (see Figure 3). At this point, item i_3 is not cut because $\delta = 2$. If the cutting process continues with horizontal cuts it will be possible to cut item i_3. A representation of the resulting patterns is given in Figure 4. Note that all meta-items containing item i_2 are created, therefore it is impossible to add it again to patterns of Figure 3-(a) and Figure 3-(b).

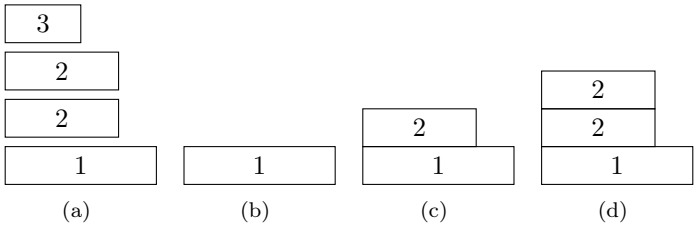

Figure 2: Vertical meta-items generation for three items: initial items (a), meta-items obtained with item 1 only ($\delta = 1$) (b), extra meta-item obtained by adding one copy of item 2 to meta-item in composed of item 1 ($\delta = 2$) (c), extra meta-item obtained by adding two copies of item 2 to meta-item composed of item 1 ($\delta = 2$) (d)

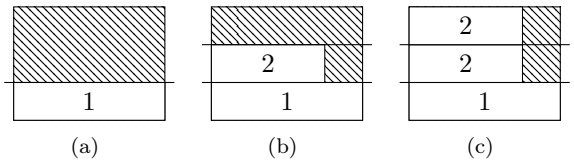

Figure 3: Vertical cutting patterns for a bin $(4, 3)$ using meta-items (b),(c) and (d) from Figure 2

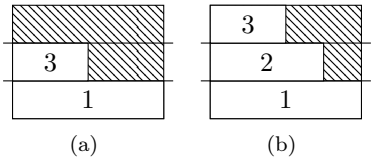

Figure 4: Complete cutting patterns for a bin $(4, 3)$ using vertical patterns from Figure 3 and adding missing item 3. Since all meta-items containing item 2 are created, it is impossible to add it again to patterns from Figure 3-(a) and Figure 3-(b)

6. Computational experiments

In this section, we report the results of experiments that we conducted on instances from the literature and on real-world data. The goal of our experiments is threefold: (*i*) to evaluate the impact of the pricing problem with partial enumeration on lower and upper bounds produced by our algorithm; (*ii*) to evaluate the quality of the solutions produced by different variants of the diving heuristic and compare them to heuristics described in Section 4.3; (*iii*) to measure the impact of re-using leftovers on real-world instances with batches.

All experiments are run using a 2.5 Ghz Haswell Intel Xeon E5-2680 with 128Go of RAM. We used CPLEX 12.6 to solve linear programs. The time limit to solve one instance using one algorithm is set to one hour.

6.1. Instances and objective functions

Two objective functions are considered. The first one is the classical cutting-stock objective (minimizing the number of plates). The second one is the total used plate width (takes into account leftovers).

We use two groups of instances. The first group has been proposed in the literature for the classical non-guillotine bin-packing problem. These instances are divided in ten classes. We denote them as $\chi \sim U[\alpha, \beta]$ if value χ is uniformly generated in interval $[\alpha, \beta]$. Six instance classes were proposed by Berkey and Wang [4] and are built as follows:

Class 1: $W = H = 10$, $w_i, h_i \sim U[1, 10]$

Class 2: $W = H = 30$, $w_i, h_i \sim U[1, 10]$

Class 3: $W = H = 40$, $w_i, h_i \sim U[1, 35]$

Class 4: $W = H = 100$, $w_i, h_i \sim U[1, 35]$

Class 5: $W = H = 100$, $w_i, h_i \sim U[1, 100]$

Class 6: $W = H = 300$, $w_i, h_i \sim U[1, 100]$

The four remaining classes with $W = H = 100$ were created by Martello and Vigo [18]. Authors divided items in types:

Type 1: $w_i \sim U[2/3W, W]$, $h_i \sim U[1, 1/2H]$

Type 2: $w_i \sim U[1, 1/2W]$, $h_i \sim U[2/3H, H]$

Type 3: $w_i \sim U[1/2W, W]$, $h_i \sim U[1/2H, H]$

Type 4: $w_i \sim U[1, 1/2W]$, $h_i \sim U[1, 1/2H]$

The four classes are:

Class 7: Type 1 with probability 70%, Type 2, 3, 4 with probability 10% each

Class 8: Type 2 with probability 70%, Type 1, 3, 4 with probability 10% each

Class 9: Type 3 with probability 70%, Type 1, 2, 4 with probability 10% each

Class 10: Type 4 with probability 70%, Type 1, 2, 3 with probability 10% each

In these instances, the total number of item copies ($\sum_{i\in\mathcal{I}} d_i$) in each instance belongs to $\{20, 40, 60, 80, 100\}$. 10 instances for each class are generated. Thus, the first group contains 500 instances. They are available at `http://or.dei.unibo.it/library/two-dimensional-bin-packing-problem`. We denote as $C - In$ the set of all instances in the first group (in classes $1 - 10$) having n items.

The second set of industrial instances is retrieved from the real-world data. In our instances $H \times W \in \{50 \times 100, 500 \times 1000, 3000 \times 6000\}$, $|\mathcal{I}| \in \{25, 50, 100\}$. The average demand of items is between 2 and 3. 15 instances were generated for each pair plate size/number of items. Thus, the second group contains 135 instances. We denote as $R - In$ the set of all instances in the second group having n items.

The industrial instances are available on the ESICUP web-site: `https://paginas.fe.up.pt/~esicup/datasets`.

6.2. Impact of the partial enumeration

Remember that the configuration of the partial enumeration in the pricing problem is characterized by four values: Δ_w^{size}, Δ_h^{size}, Δ_w^{diff}, and Δ_h^{diff}. We consider here three settings. The first one corresponds to the standard dynamic program with no enumeration: $\Delta_1 = (\Delta_w^{size} = 0, \Delta_h^{size} = 0, \Delta_w^{\text{diff}} = 0, \Delta_h^{\text{diff}} = 0)$. The second one corresponds to the enumeration of items with the same h_i for odd cutting stages and with the same w_i for even cutting stages: $\Delta_2 = (\Delta_w^{size} = 1000, \Delta_h^{size} = 1000, \Delta_w^{\text{diff}} = 1, \Delta_h^{\text{diff}} = 1)$. Our third setting corresponds to the enumeration of items with the same h_i for odd cutting stages and enumeration of items with different w_i at the second cutting stage: $\Delta_3 = (\Delta_w^{size} = 1000, \Delta_h^{size} = 1000, \Delta_w^{\text{diff}} = \infty, \Delta_h^{\text{diff}} = 1)$.

For each setting, we report in Table 1 the geometric mean of the number (in thousands) of vertices ($\mathcal{V}$), hyperarcs ($\mathcal{A}$) and the total number of hyperarc tails ($\sum |\mathcal{T}(a)|$). It can be seen from this table that the partial enumeration impact on the size of the hypergraph is very reasonable (at most 33% of increase). We do not report the construction time since it is negligible (less than 0.2 seconds on average and at most 6 seconds). Hypergraph building is a one time operation. Therefore a stricter comparison to measure hypergraph configuration has to been done when used to solve the pricing problem.

Instances	$\lvert\mathcal{V}\rvert$			$\lvert\mathcal{A}\rvert$			$\sum\lvert\mathcal{T}(a)\rvert$		
group	Δ_1	Δ_2	Δ_3	Δ_1	Δ_2	Δ_3	Δ_1	Δ_2	Δ_3
C	2.6	3.5	3.5	25.2	29.5	29.6	50.0	57.7	57.9
R	8.3	10.9	10.4	123.9	139.7	162.0	254.0	256.8	340.1

Table 1: Size of the hypergraph (geo. mean., in thousands of vertices, arcs, and tails) with different configuration of the partial enumeration

When partial enumeration is applied, many non-proper patterns are excluded from the solution space of the pricing problem. Therefore, the lower bound obtained by column generation may be improved. In the next set of experiments we computationally estimate this improvement. We have implemented four variants of column generation. The first one, denoted as cg_{mip}, uses MILP flow formulation (7)-(11) to solve the pricing problem and generates only proper columns. Thus, the best possible "proper" lower bound is obtained at the expense of a large solution time. The three other variants use dynamic programming with partial enumeration to solve the pricing problem. We denote them as cg_{Δ_1}, cg_{Δ_2}, and cg_{Δ_3} depending on the partial enumeration configuration.

The results are reported in Tables 2 and 3. In both tables, the first column reports the instance class and its total number of instances between brackets. The column *#pp* shows the number of instances solved to optimality using the fast heuristic (ea) : the obtained solution value equals to the value of the trivial "surface" lower bound. The next three columns present results for the variant cg_{mip}: number of instances not solved by (ea) for which the column generation converged within one hour, average time (t) in seconds, and the average primal-dual gap (gap) in percentage from the best known solution. Next three columns give the average gap ($gap_{\#opt}$) for other three variants of column generation. In order to have a correct comparison, averages in columns $gap_{\#opt}$ are calculated only for instances for which the variant cg_{mip} converged. Note that the column generation variants with dynamic programming converged within the time limit for all instances. Thus in columns gap_{all} and t_{all}, we give the average gap and the average solution time among all instances not solved optimally by heuristic (ea) for all hypergraph configurations. Reported times are in seconds. In Table 3 column (*#pp*) is deliberately omitted since no instances are solved by heuristic (ea).

Instances	#pp	cg_{mip}			$gap_{\#opt}$,%			gap_{all}, %			t_{all}		
		#opt	t	gap, %	cg_{Δ_1}	cg_{Δ_2}	cg_{Δ_3}	cg_{Δ_1}	cg_{Δ_2}	cg_{Δ_3}	cg_{Δ_1}	cg_{Δ_2}	cg_{Δ_3}
C-I20 (100)	63	37	85	6.94	8.63	8.38	8.29	8.63	8.38	8.29	0.1	0.1	0.2
C-I40 (100)	37	54	583	2.48	2.97	2.80	2.79	6.03	5.82	5.81	0.6	0.6	0.6
C-I60 (100)	35	33	921	1.26	1.63	1.48	1.48	3.30	3.14	3.14	1.2	1.3	1.3
C-I80 (100)	29	28	1079	1.01	1.23	1.09	1.09	2.56	2.42	2.42	2.5	2.6	2.6
C-I100 (100)	31	18	731	0.53	0.66	0.61	0.61	2.83	2.72	2.72	4.5	4.6	4.6
R-I25 (45)	37	2	1271	10.90	10.96	10.91	10.91	22.22	22.18	22.17	0.4	0.4	0.4
R-I50 (45)	34	0	-	-	-	-	-	13.85	13.82	13.82	1.9	1.9	2.0
R-I100 (45)	33	0	-	-	-	-	-	6.53	6.53	6.53	10.7	10.6	10.9

Table 2: Comparison of different column generation variants for the total number of used plates objective

Instances	cg_{mip}			$gap_{\#opt}$,%			gap_{all}, %			t_{all}		
	#opt	t	gap, %	cg_{Δ_1}	cg_{Δ_2}	cg_{Δ_3}	cg_{Δ_1}	cg_{Δ_2}	$cg_{dp\Delta_3}$	cg_{Δ_1}	cg_{Δ_2}	cg_{Δ_3}
C-I20 (100)	62	249.0	1.99	3.82	3.14	3.04	3.79	3.07	3.00	0.3	0.3	0.3
C-I40 (100)	55	1358.1	1.15	1.86	1.53	1.52	2.73	2.39	2.38	1.4	1.4	1.4
C-I60 (100)	23	1047.0	0.68	1.00	0.79	0.79	1.82	1.57	1.57	3.0	3.1	3.1
C-I80 (100)	16	506.4	0.60	0.88	0.65	0.65	1.48	1.29	1.29	5.3	5.5	5.5
C-I100 (100)	14	503.3	0.59	0.69	0.64	0.64	1.27	1.14	1.14	9.0	9.4	9.3
R-I25 (45)	20	1486.0	0.66	0.96	0.74	0.72	1.98	1.69	1.65	1.0	1.0	1.0
R-I50 (45)	4	1879.5	0.60	0.60	0.60	0.60	1.30	1.21	1.20	4.8	4.7	4.9
R-I100 (45)	0	-	-	-	-	-	0.63	0.59	0.58	26.6	25.2	27.8

Table 3: Comparison of different column generation variants for the total used plate width objective

One can see from the results in Tables 2 and 3 that the dynamic program is orders of magnitude faster than MIP for solving the pricing problem. The partial enumeration increases the running time of column generation only marginally. Moreover, this technique allows one to obtain a lower bound which is close to the "proper" lower bound, at least for the easiest instances that can be tackled by cg_{mip}. Unfortunately, we were not able to determine how close is the lower bound obtained by $cg_{dp\Delta_3}$ to the "proper" bound for larger and harder instances, as the latter bound is very time consuming to obtain. Anyway, application of the partial enumeration technique significantly increases the quality of lower bounds obtained by column generation at virtually no cost. Therefore, it offers a good trade-off between the quality of lower bounds and the column generation running time.

It can also be seen that column generation is slower for real-life instances. This is expected as the running time of the dynamic program depends on the plate size, which is larger for the instances in the second group.

In the next experiment, we estimate the impact of the partial enumeration on the pseudo-polynomial formulation (12)–(17) described in Section 3.3. By the result of Martin et al. [19], in the absence of constraints (15), this formu-

lation has the integrality property. Thus the value of its linear programming relaxation is equal to the lower bound obtained by column generation with "non-proper" columns. As we have just seen, the latter may be increased using partial enumeration. Thus, the strength of the linear programming relaxation of formulation (12)–(17) based on partly enumerated hypergraph may be improved too. Moreover, variables x may have coefficients greater than one in constraints (15) if there are hyperarcs which correspond to cutting several copies of one item, as in the case of cutting a meta-item. Thus, partial enumeration enables the possibility of adding knapsack cutting planes when solving the formulation by a MIP solver.

The results for direct solution by the CPLEX MIP solver of formulation (12)–(17) with different settings of the partial enumeration of hypergraph are shown in Tables 4 and 5. Here we first report the number $\#pp$ of instances solved by heuristic (ea). Then for each way to build the hypergraph, we report the number of remaining instances solved in one hour ($\#opt$) with MIP formulation. In the last part of the table, the average time (t) in seconds required to solve all instances is reported. If an instance is not solved within the time limit, we use the time limit value in the calculation of the average time. Reported times are in seconds.

		$\#opt$			t, sec.		
Instances (#)	$\#pp$	$MIP\Delta_1$	$MIP\Delta_2$	$MIP\Delta_3$	$MIP\Delta_1$	$MIP\Delta_2$	$MIP\Delta_3$
C-I20 (100)	63	37	37	37	0.6	0.4	0.4
C-I40 (100)	37	60	61	61	126.8	88	94.5
C-I60 (100)	35	60	62	62	292.9	167.9	173.4
C-I80 (100)	29	60	62	62	516	458.3	465.4
C-I100 (100)	31	56	56	56	752.7	655	654.4
R-I25 (45)	37	4	6	5	353	251.6	310.4
R-I50 (45)	34	3	3	2	691.1	661.8	802.1
R-I100 (45)	33	0	0	0	965.7	965.7	965.7

Table 4: Comparison of hypergraph-based MIP formulations for the 2DGCSPL with different partial enumeration for the total number of used plates objective

Instances (#)	#opt			t, sec.		
	$MIP\Delta_1$	$MIP\Delta_2$	$MIP\Delta_3$	$MIP\Delta_1$	$MIP\Delta_2$	$MIP\Delta_3$
C-I20 (100)	91	98	97	436.5	241.5	252.6
C-I40 (100)	72	73	74	1243.5	1242.3	1205
C-I60 (100)	53	57	56	1865.7	1850.9	1837.8
C-I80 (100)	48	47	47	2054.3	2061.5	2064.3
C-I100 (100)	40	42	41	2289	2227.3	2237.6
R-I25 (45)	10	17	12	2963.6	2592.5	2867.5
R-I50 (45)	0	0	1	3600	3600	3590.8
R-I100 (45)	0	0	0	3600	3600	3600

Table 5: Comparison of hypergraph-based MIP formulations for the 2DGCSPL with different partial enumeration for the total used plate width objective

According to the results, partial enumeration indeed increases the efficiency of the formulation, as more instances are solved within the time limit and the average solution time is decreased. However, the improvement is not radical. One can also notice that the instances with the total used plate width objective as well as real-life instances are significantly harder to solve.

6.3. Comparison of heuristics

In this experiments, we compare five heuristics for the 2DGCSPL:

(i) evolutionary algorithm (ea);

(ii) algorithm (iub), which combines evolutionary algorithm with list heuristics;

(iii) a diving heuristic denoted ($div_{\varnothing}$) without partial enumeration in the pricing problem and with simple evaluation of the residual problem in the diving (parameter $lastPlateOnly = true$);

(iv) a diving heuristic denoted (div) with partial enumeration Δ_3 in the pricing problem and with simple evaluation of the residual problem in the diving (parameter $lastPlateOnly = true$);

(v) a combination of the diving heuristic and the evolutionary algorithm with complete evaluation of the residual problem in the diving (parameter $lastPlateOnly = false$), which we denote as ($ediv$).

The first two heuristics are presented in Section 4.3. The other three heuristics are described in Section 5. We also consider variants with Limited Discrepancy Search for the last two heuristics and denote them as (div_{32}) and ($ediv_{32}$). In these variants, the backtrack is allowed up to the depth of 2 of the search tree and the maximum size of the tabu list is 3. This LDS parametrisation results in at most 10 dives in the search tree. Note that diving heuristics are always initialized with the solution produced by heuristic (ea).

In Tables 6 and 7, we report the average gap in percentage from the best known solution and the average time in seconds for the five heuristics and two more variants.

	ea		*iub*		$div_{\varnothing}$		*div*		div_{32}		*ediv*		$ediv_{32}$	
Instances	*gap*	*t*	*gap*	*t*	*gap*	*t*	*gap*	*t*	*gap*	*t*	*gap*	*t*	*gap*	*t*
C-I20 (100)	1.13	1	0.20	1	0.79	1	0.76	1	0.00	1	0.76	1	0.00	1
C-I40 (100)	2.63	1	1.24	3	2.23	1	2.12	1	1.53	1	2.12	1	1.53	1
C-I60 (100)	3.47	1	1.77	7	2.72	2	2.86	2	1.88	2	2.86	2	1.88	2
C-I80 (100)	4.09	3	1.62	21	2.20	3	2.38	3	1.27	5	2.38	3	1.27	5
C-I100 (100)	3.87	4	1.32	38	1.76	6	2.30	5	1.02	8	2.30	5	1.02	8
average	3.04	2	1.23	14	1.94	3	2.08	3	1.14	4	2.08	3	1.14	4
R-I25 (45)	1.11	1	1.11	1	1.11	1	1.11	1	1.11	1	1.11	1	1.11	1
R-I50 (45)	0.56	2	0.56	3	0.56	2	0.56	2	0.56	3	0.56	2	0.56	3
R-I100 (45)	0.00	8	0.00	19	0.00	12	0.00	12	0.00	18	0.00	12	0.00	18
average	0.56	4	0.56	8	0.56	5	0.56	5	0.56	8	0.56	5	0.56	8

Table 6: Comparison of heuristics for the 2DGCSPL for the total number of used plates objective

	ea		*iub*		$div_{\varnothing}$		*div*		div_{32}		*ediv*		$ediv_{32}$	
Instances	*gap*	*t*	*gap*	*t*	*gap*	*t*	*gap*	*t*	*gap*	*t*	*gap*	*t*	*gap*	*t*
C-I20 (100)	3.47	1	2.11	1	2.09	1	2.26	1	1.73	1	1.87	1	1.64	1
C-I40 (100)	3.64	1	1.79	4	1.59	2	1.42	2	0.55	4	0.86	3	0.37	8
C-I60 (100)	3.68	1	1.80	13	1.13	3	1.09	4	0.39	9	0.55	8	0.21	27
C-I80 (100)	4.14	2	1.94	32	0.99	6	0.78	6	0.19	17	0.36	19	0.07	79
C-I100 (100)	4.47	4	1.84	65	0.75	10	0.56	10	0.15	28	0.32	36	0.02	170
average	3.88	2	1.90	23	1.31	5	1.22	5	0.60	12	0.79	14	0.46	57
R-I25 (45)	2.66	1	2.56	2	1.73	2	1.40	2	0.91	5	1.17	2	0.75	7
R-I50 (45)	1.93	2	1.83	16	1.02	8	0.54	10	0.17	53	0.33	19	0.01	112
R-I100 (45)	1.51	8	1.37	175	0.61	48	0.30	55	0.08	369	0.18	155	0.00	1224
average	2.03	4	1.92	65	1.12	20	0.75	23	0.38	143	0.56	59	0.26	448

Table 7: Comparison of heuristics for the 2DGCSPL for the total used plate width objective

As observed from Table 6, diving heuristics with LDS led to the best results for the total number of used plates objective: they produced solutions with the best average quality within a few seconds. For group of instances R, a good quality solution is almost always obtained by fast heuristic (*ea*).

Nevertheless, instances with the total used plate width objective are more interesting for us, as they represent batches in the real problem to solve. From Table 7 one can see that diving algorithms clearly outperform the first two heuristics. Heuristic (*ea*) is the fastest but also produces solutions of the worst quality. Heuristic (*iub*) improves the solution quality at the expense of much larger running time. However, it struggles with real-life instances, as the solu-

tion improvement over (ea) for this instances is very small. Diving algorithms ($div_{\varnothing}$) and (div) significantly outperform heuristic (iub) both in terms of running time and solution quality. The partial enumeration technique increases the effectiveness of the diving heuristic for most instances at a very small cost. This technique is especially useful for large instances with the total used plate width objective. The combination ($ediv$) of the diving heuristic and the evolutionary algorithm further improves the quality of the obtained solutions at a cost of a reasonable increase of running time. The best solutions on average are obtained by diving heuristics with Limited Discrepancy Search. However, the running time of these heuristics is quite long especially for large real-life instances. Thus, to our opinion, the heuristics (div) and ($ediv$) offer the best tradeoff between solution quality and running time.

In the industrial context, the 2DGCSPL is solved iteratively for different batches of the global problem. To validate our methodology, we consider now real-world instances with batches and plate dimension $H \times W = (3000 \times 6000)$. Each instance is composed of 10 or 15 batches, each batch b is composed of $|\mathcal{I}_b| \in \{100, 150\}$ items. As previously, average demand of items is between 2 and 3. For each class, we retrieved 25 instances. The class named $LmIn$ contains instances with m batches, each of which having n items. All these instances are are available on the ESICUP web-site: `https://paginas.fe.up.pt/~esicup/datasets`.

In Table 8, we report the results obtained by the same heuristics tested before. Here we do not report results only for heuristic ($div_{\varnothing}$) as it was shown to be dominated by (div). We also do not report results obtained by div_{32} and $ediv_{32}$ since required computation time is very long and obtained solutions are only slightly better than the solutions obtained after only one dive. For each heuristic, we show the solution value obtained which equals to the average number of plates needed to cut items from all batches, as well as the average solution time in minutes. We also give the average lower bound (lb) on the optimal value. For each instance, this value is obtained by iteratively computing the rounded-up column generation lower bound for each batch and determining the length of the leftover plate for the next bin based on this bound. To measure the impact of using leftover plates, in column "w/o lo" we give the average of the best solution values obtained by our algorithms for the problem variant in which the batch leftovers cannot be reused in the next batch.

			Solution value				t, min.			
Instances	w/o lo	lb	ea	iub	div	$ediv$	ea	iub	div	$ediv$
L10I100	130.7	123.3	126.5	126.4	124.4	124.4	1	32	24	48
L10I150	199.6	191.5	195.5	195.3	192.8	192.5	4	144	83	194
L15I100	203.2	191.7	196.4	196.2	193.5	193.2	2	50	36	72
L15I150	289.6	277.3	283.3	282.9	279.2	279.0	6	208	126	291

Table 8: Comparison of heuristics on the real-world instances with batches

As observed in Table 8, keeping a leftover plate for the next batch allows one to save up to 10.6 plates or 3.7% of plates on average. Again, heuristic (*ea*) is the fastest. The more expensive heuristic (*iub*) improves on (*ea*) only marginally. Much better results are obtained by diving heuristics. The standard diving heuristic (*div*) saves up to 4.1 plates or 1.4% of plates on average. The extended diving heuristic (*ediv*) saves up to 4.3 plates or 1.5% of plates on average. Moreover the gap with the lower bound is at most 1.9 plates or 0.7% of plates on average using (*div*). For the (*ediv*) this drops to 1.7 plates or 0.6% on average. In our opinion heuristic (*div*) offers the best solution quality – running time trade-off. Even if its running time reaches 2 hours for the largest instances, its application in practice is still realistic. These instances correspond to a one day planning horizon. Therefore, spending two hours to obtain a solution seems to be reasonable.

7. Conclusions

In this work we studied the two-dimensional guillotine cutting-stock problem with leftovers (2DGCSPL) which is solved consecutively when the set of items is partitioned in batches. Given the difficulty and large size of real-life instances, we considered the special case with restricted cuts.

To solve the 2DGCSPL, we have proposed column generation based diving heuristics. An originality of our work is that these heuristics work with non-proper cutting patterns. This variant simplifies the pricing problem but makes it more difficult to obtain feasible solutions for the 2DGCSPL. We proposed several ways to overcome this difficulty, including combination with an evolutionary algorithm and a partial enumeration technique. The latter reduces the number of generated non-proper cutting patterns, tightens the column generation lower bounds and improves the quality of solutions obtained by the diving heuristics.

The computational experiments on the literature and real-life instances revealed that the proposed diving heuristics outperformed significantly the constructive and evolutionary heuristics. The largest improvements are achieved on large instances, as well as on real-life instances and for the instances with the total width objective. The experiments on real-life production plant instances with batches showed that our heuristics run in a reasonable time and allow the decision maker to save up to 1.5% of raw material on average.

Our diving heuristic with non-proper columns is generic and could be applied to other two-dimensional cutting-stock problems. It would be interesting to see how the heuristic running time evolves when considering instances with a different number of stages or with non-restricted cuts. Generalisation to the case with plate defects is especially useful for practical purposes.

We also hope that the publication of real-life instances will inspire more research on this family of problems and will allow the researchers to perform a fair computational comparison between future algorithmic approaches.

Acknowledgment

Experiments presented in this paper were carried out using the PLAFRIM experimental testbed, being developed under the Inria PlaFRIM development action with support from Bordeaux INP, LABRI and IMB and other entities: Conseil Régional d'Aquitaine, Université de Bordeaux, CNRS and ANR in accordance to the programme d'investissements d'Avenir (see `https://www.plafrim.fr/`).

[1] Alvarez-Valdes, R., Parajon, A., Tamarit, M. J., 2002. A computational study of lp-based heuristic algorithms for two-dimensional guillotine cutting stock problems. OR Spectrum 24 (2), 179–192.
URL `http://dx.doi.org/10.1007/s00291-002-0093-3`

[2] Andrade, R., Birgin, E., Morabito, R., 2016. Two-stage two-dimensional guillotine cutting stock problems with usable leftover. International Transactions in Operational Research 23 (1-2), 121–145.
URL `http://dx.doi.org/10.1111/itor.12077`

[3] Beasley, J. E., 1985. Algorithms for unconstrained two-dimensional guillotine cutting. Journal of the Operational Research Society 36 (4), 297–306.
URL `http://dx.doi.org/10.1057/jors.1985.51`

[4] Berkey, J. O., Wang, P. Y., 1987. Two-dimensional finite bin-packing algorithms. Journal of the Operational Research Society 38 (5), 423–429.
URL `http://dx.doi.org/10.1057/jors.1987.70`

[5] Cintra, G., Miyazawa, F., Wakabayashi, Y., Xavier, E., 2008. Algorithms for two-dimensional cutting stock and strip packing problems using dynamic programming and column generation. European Journal of Operational Research 191 (1), 61 – 85.
URL `http://dx.doi.org/10.1016/j.ejor.2007.08.007`

[6] Clautiaux, F., Sadykov, R., Vanderbeck, F., Viaud, Q., 2017. Combining dynamic programming with filtering to solve a four-stage two-dimensional guillotine-cut bounded knapsack problem. Tech. Rep. hal-01426690, HAL Inria.

[7] Dolatabadi, M., Lodi, A., Monaci, M., 2012. Exact algorithms for the two-dimensional guillotine knapsack. Computers & Operations Research 39 (1), 48–53.
URL `http://dx.doi.org/10.1016/j.cor.2010.12.018`

[8] Dusberger, F., Raidl, G. R., 2014. A variable neighborhood search using very large neighborhood structures for the 3-staged 2-dimensional cutting stock problem. In: Blesa, M. J., Blum, C., Voß, S. (Eds.), Hybrid Metaheuristics: 9th International Workshop, HM 2014, Hamburg, Germany, June 11-13, 2014. Proceedings. Springer International Publishing, Cham, pp. 85–99.
URL `http://dx.doi.org/10.1007/978-3-319-07644-7_7`

[9] Dusberger, F., Raidl, G. R., 2015. Solving the 3-staged 2-dimensional cutting stock problem by dynamic programming and variable neighborhood search. Electronic Notes in Discrete Mathematics 47, 133 – 140.
URL http://dx.doi.org/10.1016/j.endm.2014.11.018

[10] Fleszar, K., 2013. Three insertion heuristics and a justification improvement heuristic for two-dimensional bin packing with guillotine cuts. Computers & Operations Research 40 (1), 463 – 474.
URL http://dx.doi.org/10.1016/j.cor.2010.12.018

[11] Furini, F., Malaguti, E., Durán, R. M., Persiani, A., Toth, P., 2012. A column generation heuristic for the two-dimensional two-staged guillotine cutting stock problem with multiple stock size. European Journal of Operational Research 218 (1), 251 – 260.
URL http://dx.doi.org/10.1016/j.ejor.2011.10.018

[12] Furini, F., Malaguti, E., Thomopulos, D., 2016. Modeling two-dimensional guillotine cutting problems via integer programming. INFORMS Journal on Computing 28 (4), 736–751.
URL http://dx.doi.org/10.1287/ijoc.2016.0710

[13] Gilmore, P. C., Gomory, R. E., 1965. Multistage cutting stock problems of two and more dimensions. Operations research 13 (1), 94–120.
URL http://dx.doi.org/10.1287/opre.13.1.94

[14] Hadjiconstantinou, E., Iori, M., 2007. A hybrid genetic algorithm for the two-dimensional single large object placement problem. European Journal of Operational Research 183 (3), 1150 – 1166.
URL http://dx.doi.org/10.1016/j.ejor.2005.11.061

[15] Lodi, A., Martello, S., Vigo, D., 2002. Recent advances on two-dimensional bin packing problems. Discrete Applied Mathematics 123 (1–3), 379 – 396.
URL http://dx.doi.org/10.1016/S0166-218X(01)00347-X

[16] Lodi, A., Monaci, M., 2003. Integer linear programming models for 2-staged two-dimensional knapsack problems. Mathematical Programming 94 (2-3), 257–278.
URL http://dx.doi.org/10.1007/s10107-002-0319-9

[17] Macedo, R., Alves, C., Valério de Carvalho, J. M., 2010. Arc-flow model for the two-dimensional guillotine cutting stock problem. Computers & Operations Research 37 (6), 991–1001.
URL http://dx.doi.org/10.1016/j.cor.2009.08.005

[18] Martello, S., Vigo, D., 1998. Exact solution of the two-dimensional finite bin packing problem. Management Science 44 (3), 388–399.
URL http://dx.doi.org/10.1287/mnsc.44.3.388

[19] Martin, R. K., Rardin, R. L., Campbell, B. A., 1990. Polyhedral characterization of discrete dynamic programming. Operations Research 38 (1), 127–138.
URL http://dx.doi.org/10.1287/opre.38.1.127

[20] Pessoa, A., Sadykov, R., Uchoa, E., Vanderbeck, F., 2017. Automation and combination of linear-programming based stabilization techniques in column generation. INFORMS Journal on Computing (Forthcoming).

[21] Polyakovsky, S., M'Hallah, R., 2009. An agent-based approach to the two-dimensional guillotine bin packing problem. European Journal of Operational Research 192 (3), 767 – 781.
URL http://dx.doi.org/10.1016/j.ejor.2007.10.020

[22] Puchinger, J., Raidl, G. R., 2007. Models and algorithms for three-stage two-dimensional bin packing. European Journal of Operational Research 183 (3), 1304–1327.
URL http://dx.doi.org/10.1016/0377-2217(95)00128-X

[23] Puchinger, J., Raidl, G. R., Koller, G., 2004. Solving a real-world glass cutting problem. In: Gottlieb, J., Raidl, G. R. (Eds.), Evolutionary Computation in Combinatorial Optimization: 4th European Conference, EvoCOP 2004, Coimbra, Portugal, April 5-7, 2004. Proceedings. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 165–176.
URL http://dx.doi.org/10.1007/978-3-540-24652-7_17

[24] Russo, M., Sforza, A., Sterle, C., 2014. An exact dynamic programming algorithm for large-scale unconstrained two-dimensional guillotine cutting problems. Computers & Operations Research 50, 97 – 114.
URL http://dx.doi.org/10.1016/j.cor.2014.04.001

[25] Sadykov, R., Vanderbeck, F., Pessoa, A., Tahiri, I., Uchoa, E., 2016. Primal heuristics for branch-and-price: the assets of diving methods. Tech. Rep. hal-01237204, HAL Inria.

[26] Silva, E., Alvelos, F., Valério de Carvalho, J. M., 2010. An integer programming model for two-and three-stage two-dimensional cutting stock problems. European Journal of Operational Research 205 (3), 699–708.
URL http://dx.doi.org/10.1016/j.ejor.2010.01.039

[27] Valério de Carvalho, J. M., 1999. Exact solution of bin-packing problems using column generation and branch-and-bound. Annals of Operations Research 86, 629–659.
URL http://dx.doi.org/10.1023/A:1018952112615

[28] Vanderbeck, F., 2001. A nested decomposition approach to a three-stage, two-dimensional cutting-stock problem. Management Science 47 (6), 864–879.
URL http://dx.doi.org/10.1287/mnsc.47.6.864.9809